

D.D.S.R

Relatório nº3 **- Modelação e simulação de múltiplos acessos e links ponto-a-ponto**

**Curso:**METI **Turno:** 3ª feira 15:00 - 16:30)

**Grupo: 8**

|  |
| --- |
| **Trabalho realizado por:** |
| Ruben Condesso, nº 81969 André Mendes, nº78079 |

# Exercício 1.a:

# No cálculo da utilização média, segundo o protocolo de *Aloha*, tivemos em consideração uma cadeia de *Markov* de N estados, onde em cada um desses estados correspondia ao número de utilizadores que se encontravam no estado ativo.

# A utilização média deste protocolo é dada por: . É importante salientar, que π representa o vetor que contém as *limiting state probabilities* e S(i) contém o *conditional* *throughput* ( ou seja, é utilização condicionada do estado i), sendo este dado por: .

# O vetor *limiting state state probabilities*, que contém as probabilidades estacionárias para cada estado i, é dado por: , onde: 1 é um vetor com N elementos; P é a matriz de probabilidades de transição de estados; E é uma matriz NxN com todos os elementos iguais a 1; e I é a matriz identidade NxN.

# Relativamente à matriz P, seguimos o seguinte raciocínio recorrendo aos *slides* dados na aula:

* *Para* ***j ≤ i – 2****:* Transições para dois ou mais estados abaixo não são possíveis, sendo que só é possível transmitir um único pacote com sucesso em cada *slot*. Ou seja,
* *Para* ***j = i - 1***: Transições para o estado imediatamente anterior só são possíveis quando existe uma transmissão com sucesso e não são gerados pacotes dos utilizadores em estado inativo.

1. Probabilidade de existir uma transmissão bem sucedida: ;
2. Probabilidade de não ser gerado qualquer pacote: ;
3. Probabilidade da transição de estado: ;

* *Para* ***j = i****:* Não existem transições de estado se não forem gerados novos pacotes e forem transmitidos 0 ou mais que 1 pacotes; ou se for gerado exatamente um pacote e for transmitido 1 pacote.

1. *Probabilidade de não serem gerados novos pacotes e forem transmitidos 0 ou mais que 1 pacotes: ;*
2. *Probabilidade de ser gerado exatamente um pacote e for transmitido 1 pacote: ;*
3. *Probabilidade da transição de estado:*

* *Para* ***j > i****:* Aumento de utilizadores ativos, tal é originado se forem gerados i\*j pacotes e se forem feitas 0 ou mais transmissões para o canal; ou se forem gerados (i+1)\*j pacotes e se for transmitido apenas um pacote.

1. Probabilidade de serem gerados i\*j pacotes e serem feitas 0 ou mais transmissões para o canal: ;
2. Probabilidade de serem gerados (i+1)\*j pacotes e ser transmitido apenas um pacote: ;
3. Probabilidade da transição de estado:

# Para o calculo deste valor teórico, da utilização média segundo o protocolo *Aloha*, foram desenvolvidas as 4 funções explicadas anteriormente, segundo um código matlab: *conditionalThroughput* - função que gera a taxa de transferência condicional, *S(i)*; *matrizTransicaoEstados* - função que gera a matriz de transição de probabilidades, *P*; *matrizPI* - função que gera o vetor *limiting state probabilities*, *PI*; *theoreticalThroughput* - função que gera a taxa de transferência teórica (*throughput*).

# 

**Figura** 2: *Throughput* teórico para p=0.5; σ=0.4;N=10

**Figura** 1: *Throughput* teórico para p=0.4; σ=0.5;N=10

# 

# 

**Figura 4**: *Throughput* teórico para p=0.5; σ=0.4;N=20

**Figura 3**: *Throughput* teórico para p=0.4; σ=0.5;N=20

# 

# 

**Figura 6**: *Throughput* teórico para p=0.6; σ=0.3;N=10

**Figura 5**: *Throughput* teórico para p=0.3; σ=0.6;N=10

# Com o auxílio das figuras ilustradas acima, podemos concluir que à medida que N tende para infinito, o *throughput* tende para 0, e para valores maiores de p e inversamente valores menores de sigma, o *throughput* tende também para 0.

# Exercício 1.b:

# Para simular o processo do protocolo de *Aloha*, desenvolvemos as seguintes funções:

# Função *actualizacaoEstados*: Tem como objetivo, dado um certo *p*, um certo *sigma* e um certo vetor de utilizadores, *users*, atualizar o estado de cada utilizador. Se o utilizador *i* estiver no estado ativo é feita uma *bernoulli* de *p*, e caso o resultado seja 1, é alterado o estado do mesmo para 2 (estado de espera); se o utilizador *i* estiver no estado inativo, é feita uma *bernoulli* de *sigma*, e caso o estado seja 1, é alterado para 1 o estado do utilizador, ou seja, para o estado ativo. É retornado o vetor com os estados atualizados dos utilizadores;

# Função *mudarEstado*: A sua função consiste em alterar o estado de um utilizador *i*: é recebido uma variável, *estado*, que contém o valor do estado a alterar e o vetor *users*. Se *estado* for igual a 1, verifica-se se o utilizador *i* está no estado 2, ou seja, queria enviar um pacote mas houve uma colisão, logo têm de ficar no estado ativo para tentar enviar posteriormente, desta forma, o seu estado passa a 1; se o valor de *estado* não for 1, verifica-se se o utilizador *i* estava à espera de enviar algum pacote, e caso esteja, o seu estado passa a 1 porque conseguiu enviar com sucesso. É retornado o vetor com a mudança dos estados dos utilizadores ;

# Função *numeroColisoes*: Têm como finalidade devolver o número de utilizadores à espera de enviar pacotes, para isso percorre o vetor de utilizadores, *users*, que recebe como input, e verifica o estado de cada um. Se o estado do utilizador *i* for igual a 2, é aumentado o contador;

# Função *slottedAlohaSimulation*: Esta função faz a simulação de *Aloha* propriamente dita, recorrendo às funções anteriores. Recebe como input o valor de *p*, *sigma*, o número de utilizadores (*Nusers*) e o número de slots (*Nslots*). É feito um ciclo for até ao número de slots dado, e em cada iteração é chamada a função *actualizacaoEstados*, que irá fazer a atualização dos estados dos utilizadores, dado os valores de *p* e *sigma*, recorrendo a *bernoulli*. Depois verifica se houve colisões. Caso tenha havido é feita a mudança de estado dos utilizadores para 1, e caso não tenha havido é feita para 0, como foi explicado em cima. Para cada pacote enviado com sucesso é incrementado o valor da variável que representa o número de sucessos. É retornado esse valor a dividir pelo número de slots.;

# Por fim, criamos a função *ex1* que apenas faz o display do valor teórico calculado e o valor prático gerado, chamando assim, as funções *theoreticalThroughput* e *slottedAlohaSimulation*, respetivamente. De seguida, será ilustrado essa comparação:

# 

**Figura 7** - Comparação entre o valor teórico calculado e o valor das simulações de *Aloha*, para *p*=0.3;*sigma*=0.4;*Nusers*=5;*Nslots*=100

**Figura 8** - Comparação entre o valor teórico calculado e o valor das simulações de *Aloha*, para *p*=0.3;*sigma*=0.4;*Nusers*=10;*Nslots*=100

# 

# 

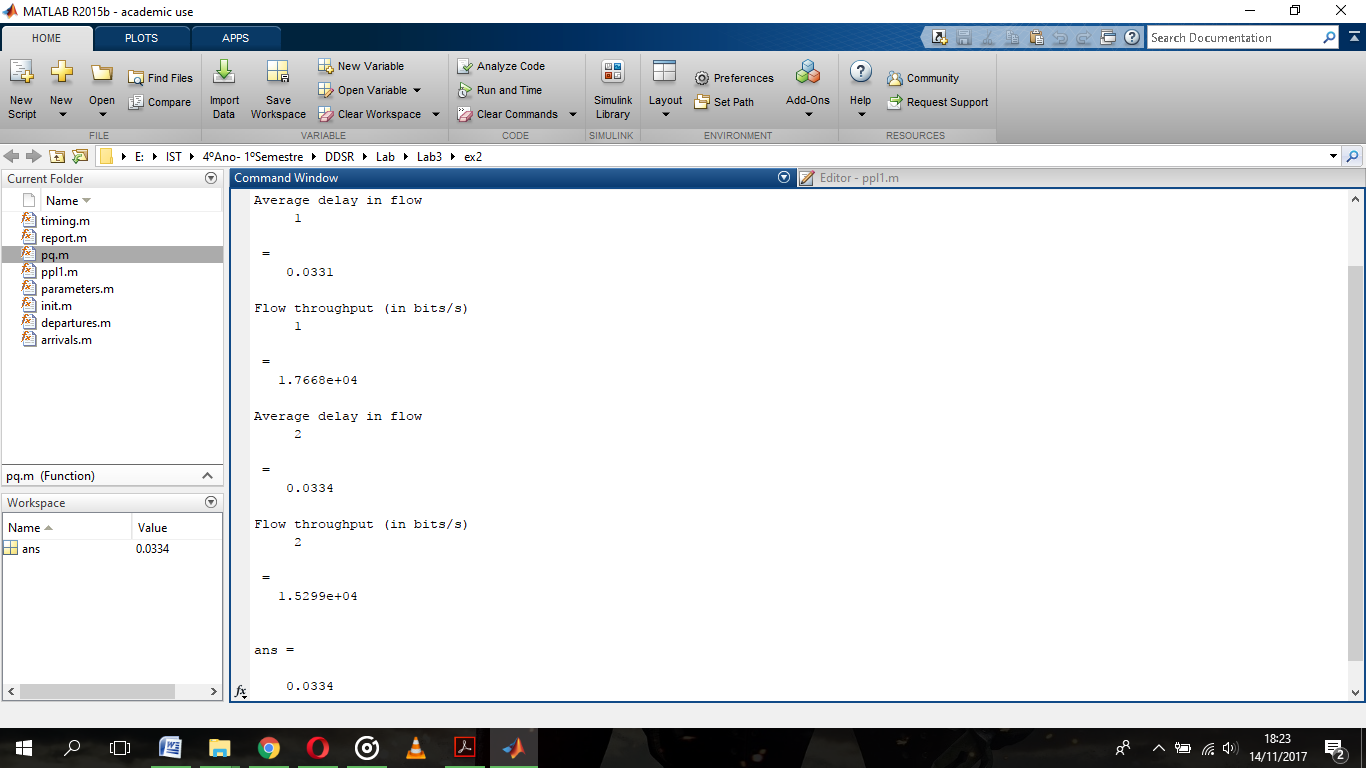
**Figura 9** - Comparação entre o valor teórico calculado e o valor das simulações de *Aloha*, para *p*=0.3;*sigma*=0.4;*Nusers*=15;*Nslots*=100

**Figura 10** - Comparação entre o valor teórico calculado e o valor das simulações de *Aloha*, para *p*=0.3;*sigma*=0.4;*Nusers*=15;*Nslots*=200,300,400

# Através das figuras ilustradas em cima, podemos retirar várias conclusões válidas, dado os valores de *p*, *sigma*, *Nusers* e *Nslots* apresentados: para um número de utilizadores reduzido, o valor prático gerado é superior ao valor teórico, tal pode de ser na figura 7. Se aumentarmos o número de utilizadores, o resultado obtido através da simulação vai se aproximando do valor teórico, onde para 10 utilizadores esse valor já é inferior ao teórico, como está representado na figura 8. Por isso será entre 5 e 10 utilizadores que o valor da simulação mais se aproxima do valor teórico calculado. À medida que se aumenta o número de utilizadores, o valor da simulação vai tendendo para 0, como seria de esperar, na figura 9 verifica-se que é praticamente 0.

# O número de *slots*, para valores superiores a 100 deixa de ter grande impacto no resultado obtido através da simulação, como pode ser verificado na figura 10, desta forma foi sempre usado o valor 100 nas restantes simulações, pois era o valor mais adequado.

# Exercício 2:

 Para este exercício, começamos por alterar, na função *parameters*, o valor da *source type* para 1 em ambos os *flows*, onde 1 representa "*Poisson arrivals and exponentially distributed sizes*", e alteramos ainda o valor da prioridade do *flow* 2 para 1, de modo a ambos terem a mesma prioridade. Desta forma, já estaria implementado um *link* ponto-a-ponto, com o protocolo FIFO, para os dois fluxos que tinham chegadas de *Poisson* e tamanhos de pacotes exponencialmente distribuídos.

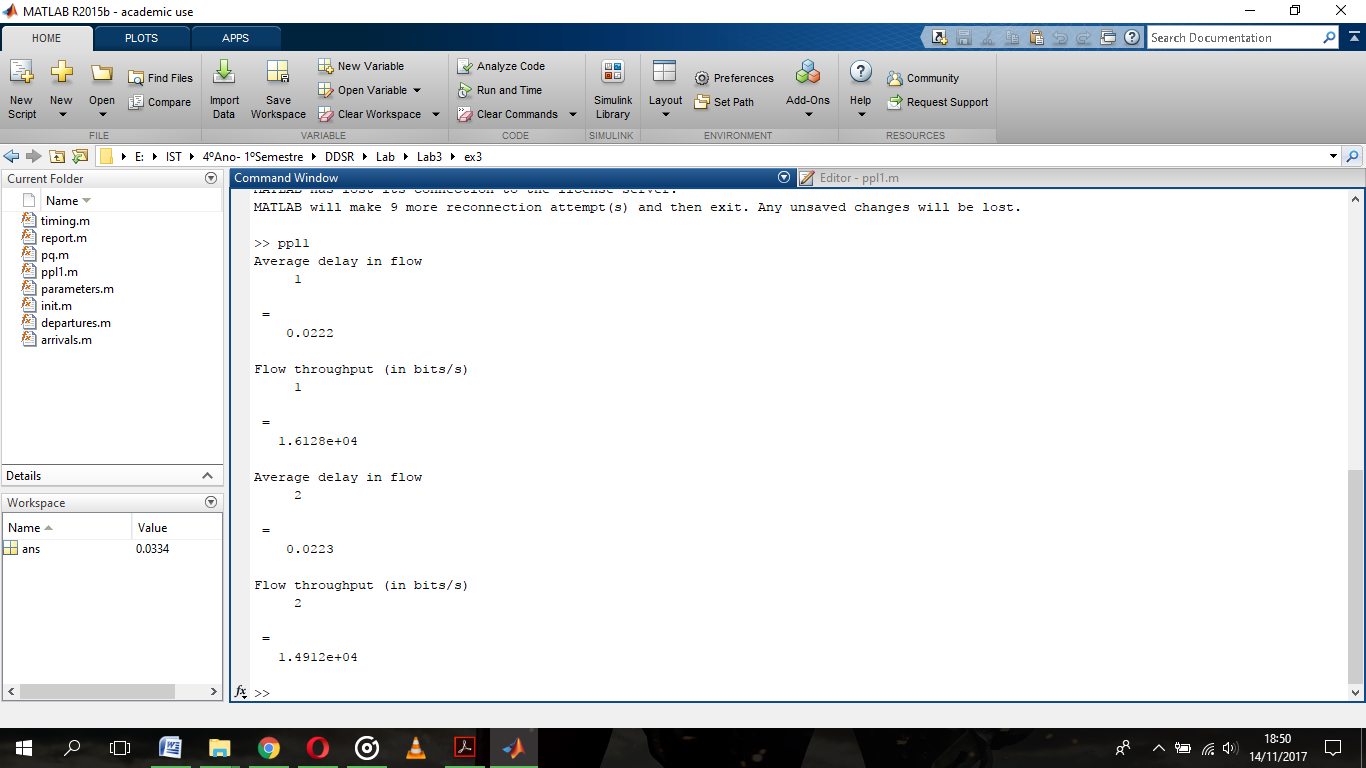
Correndo a função *ppl1*, chegamos à seguinte simulação:

**Figura 11** - *link* ponto-a-ponto, com o protocolo FIFO, para os dois fluxos que tinham chegadas de *Poisson* e tamanhos de pacotes exponencialmente distribuídos

Relativamente ao cálculo do valor teórico, considerou-se e tendo cada fluxo, um comportamento semelhante ao sistema M/M/1, temos então que o atraso médio teórico será , que é o valor muito aproximado do valor prático gerado na figura 11.

# Exercício 3:

Agora relativamente este exercício, as mudanças feitas voltaram a ser na função *parameters*, mais concretamente no *array flows*: o valor do *source type* colocou-se a 2 e o valor da prioridade pôs-se a 1, em ambos os *flows*. Agora estas alterações irão fazer com que os dois *flows* tenham chegadas de *Poisson* e pacotes de tamanho fixo, ou seja, irão ter um comportamento semelhante ao sistema M/D/1.

 Com isto, originámos então a seguinte simulação:

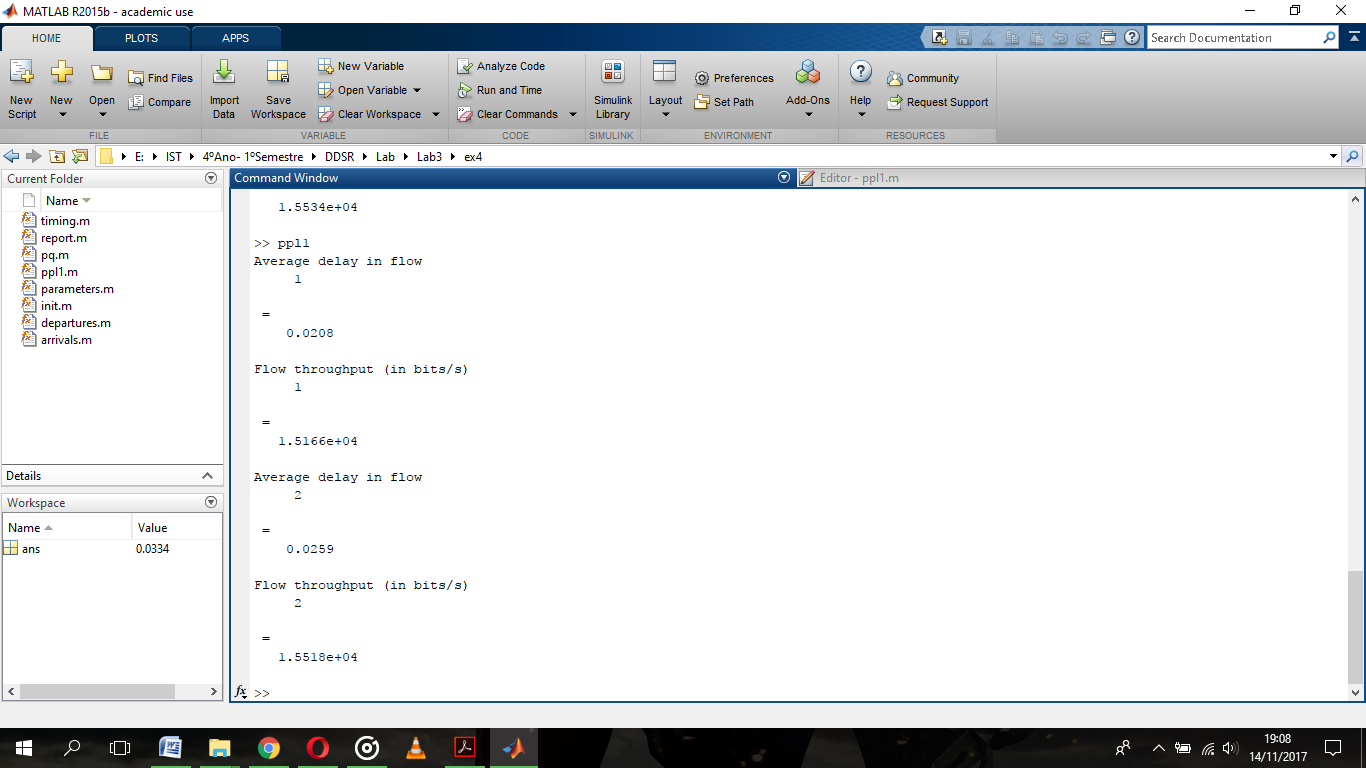
**Figura 12** - *link* ponto-a-ponto, com o protocolo FIFO, para os dois fluxos que tinham chegadas de *Poisson* e tamanhos de pacotes de tamanho fixo

Para calcular o valor teórico, e tendo por base o sistema M/D/1, chegamos ao seguinte valor para o atraso médio teórico: Comparativamente aos valores gerados, que estão ilustrados na figura 12, há uma grande aproximação de resultados.

# Exercício 4:

Para este exercício, com o intuito de simular um *link* ponto-a-ponto com dois *flows* de prioridades diferentes, com chegadas de *Poisson* e pacotes de tamanho fixo, alterou-se na função *parameters*, no *array flows* o valor da prioridade de cada *flow*, ficando o primeiro com 1 e o segundo com 2, ou seja, ficando o *flow* 1 com uma prioridade mais baixa relativamente ao *flow* 2. Os valores do *source type* mantiveram-se a 2.

Com estas alterações, gerámos a seguinte simulação:



**Figura 13** - *link* ponto-a-ponto, com o *strict priority scheduling*, para os dois fluxos que tinham chegadas de *Poisson* e tamanhos de pacotes de tamanho fixo, com prioridades diferentes

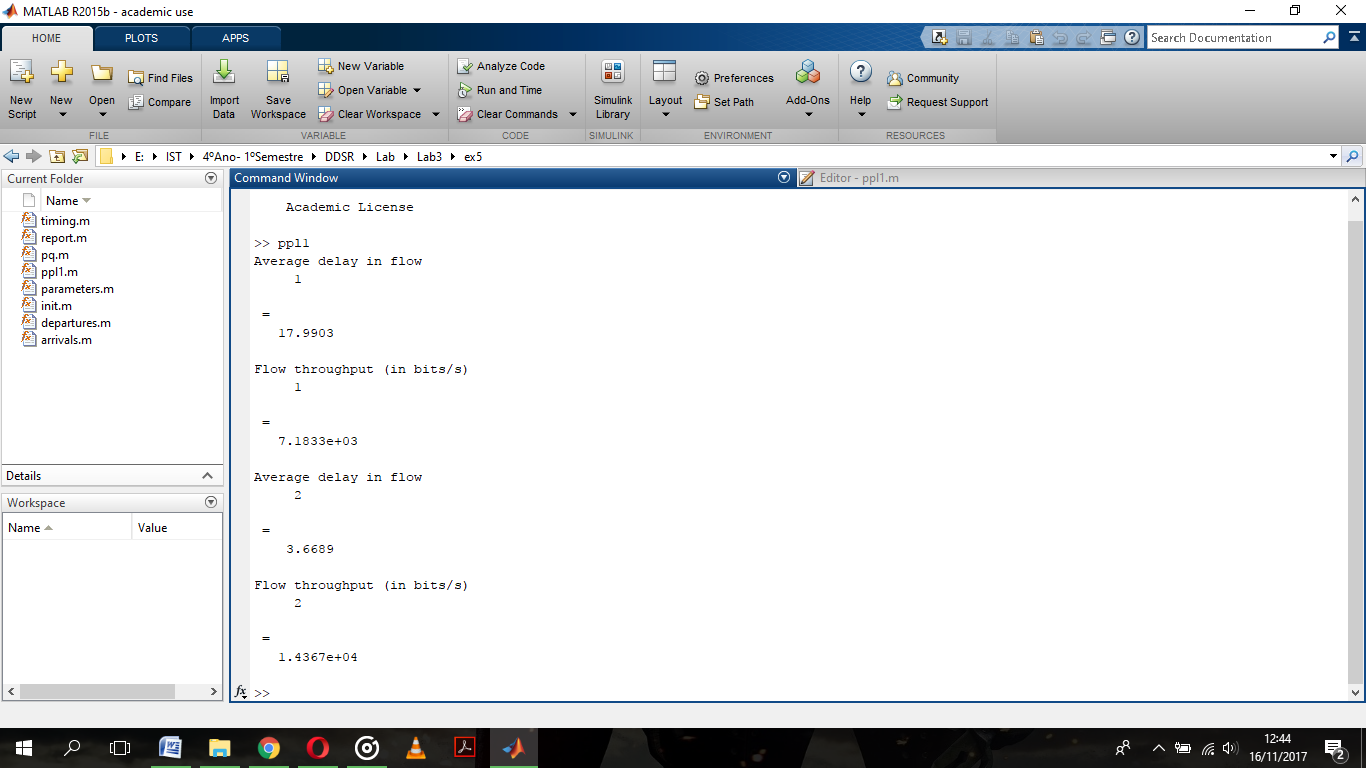
Em relação ao valor teórico, esta simulação têm um comportamento semelhante, neste caso, ao sistema M/G/1 com prioridades, e desde logo o valor teórico do atraso médio é dado pela fórmula:

Como podemos observar ao comparar os valores teóricos com os valores gerados na figura 13, não há uma aproximação entre eles.

# Exercício 5:

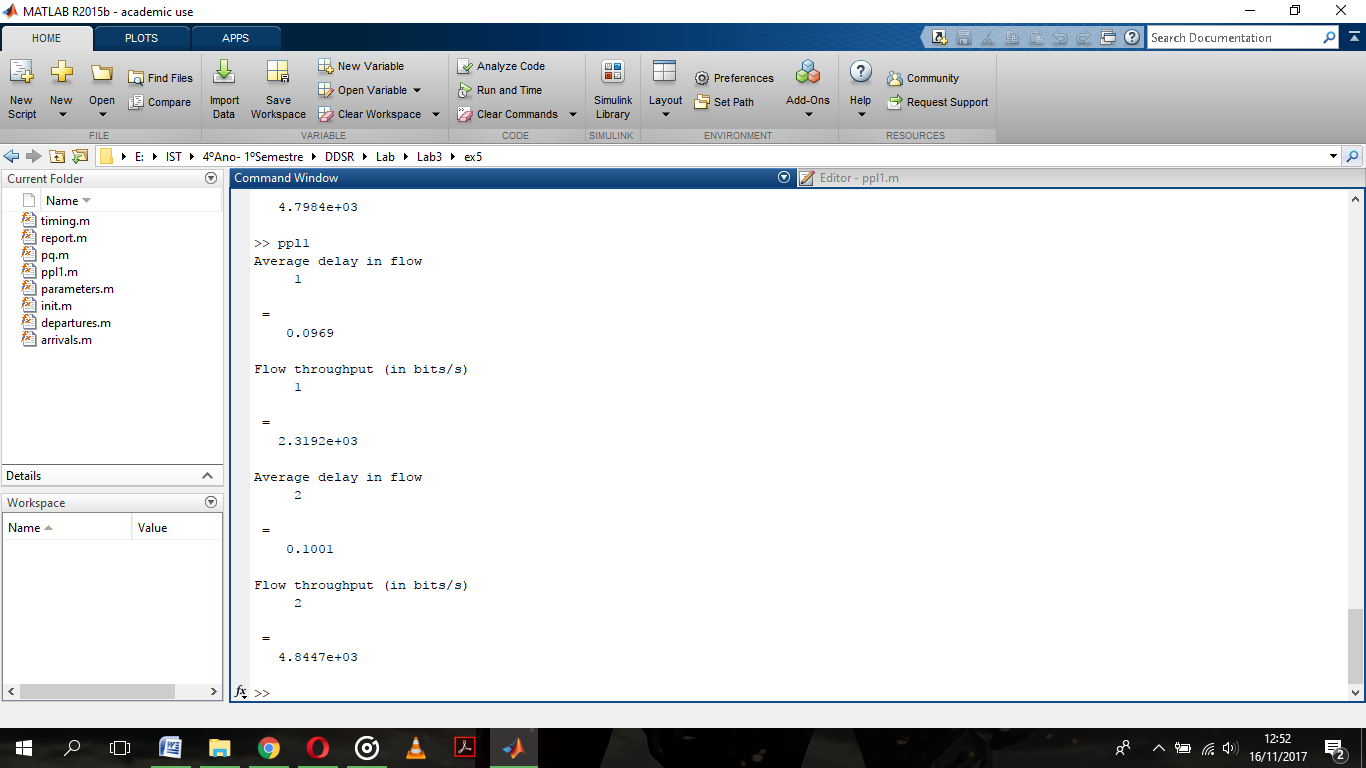
No que diz respeito a este exercício, é proposto implementar o *Deficit Round Robin scheduling*. Para esse efeito começamos por alterar a função *parameters*, adicionando mais um campo ao *array flows*, parâmetro esse que passa a representar o *quantum* de cada fluxo e desta forma é possível saber qual é o *credit threshold* de cada fluxo. De seguida, na função *init* inicializámos a variável *currentQueue* a 1 e atualizamos os créditos disponíveis para o respetivo fluxo, sendo a variável *Credits* que representa isso. Finalmente, na função *pq* alterámos o código de modo a haver uma seleção de fila, com base no funcionamento do *Deficit Round Robin*, ou seja, com base no crédito existente em cada fila na altura e tendo em conta se a fila está ou não vazia. Há que referir também que o mecanismo de prioridades implementado no exercício anterior foi igualmente utilizado para haver uma separação de tráfego dos dois fluxos para filas diferentes.

Fizemos duas simulações para verificar o funcionamento do *Deficit Round Robin scheduling*, e demos certos valores ao array *flows* da função *parameters* de modo a criar uma saturação da rede e assim conseguir retirar conclusões válidas das simulações. Para a 1º simulação usámos: 250 e 500 de *credit threshold* para o fluxo 1 e 2, respetivamente; e o mesmo tamanho para cada pacote, 1000 bits, tamanho fixo. Originámos a seguinte simulação:



**Figura 14** - *Deficit Round Robin*, para lambda1=lambda2=16; 250 e 500 de crédito para fluxo 1 e fluxo 2; e pacotes com 1000 bits

Para a 2º simulação, usámos 100 e 200 de *credit threshold* para o fluxo 1 e 2, respetivamente; pacotes com 150 bits para o fluxo 1 e com 300 bits para o fluxo 2 de tamanho fixo; . Originámos a seguinte simulação:



**Figura 15** - *Deficit Round Robin*, para lambda1=lambda2=16; 100 e 200 de crédito para fluxo 1 e fluxo 2; e pacotes com 150 bits e 300 bits para fluxo 1 e 2, respetivamente

Relativamente ao valor teórico do *throughput*, para o algoritmo de *Deficit Round Robin*, é dado pela seguinte expressão: . Posto isto, é de esperar que em ambas as simulações o *throughput* relativo ao fluxo 2 seja duas vezes o do fluxo 1, sendo que o crédito do fluxo 2 é igual ao dobro do crédito do fluxo 1.

Pegando nesta conclusão, e verificando o *display* das duas simulações, permite-nos concluir que o *throughput* depende do crédito fixado para cada fluxo e não do tamanho fixo de cada pacote.

**Anexo exercício 1:**

*(Para o exercício 1.a)*

function [ Si ] = conditionalThroughput( prob, i)

% Performance Slotted ALOHA - Conditional Throughtput S(i)

Si = i\*(prob)\*((1-prob)^(i-1));

end

function [f] = Bernoulli(p)

% funcao bernoulli - retorna 1 com probabilidade p

if(rand < p)

f = 1;

else

f= 0;

end

end

function [ f ] = matrizTransicaoEstados( prob, sigma, Nusers )

% Performance Slotted ALOHA - Matriz de Transição de Estados

% Matriz de Transicao dos Estados

matrizP= magic(Nusers); % Matriz N por N construída a partir dos inteiros 1 a n^2 com soma % de colunas e linhas iguais

i=1;

j=1;

while i<=Nusers % States of Markov Chain = Number of ACTIVE users

if j>Nusers

i=i+1;

j=1;

continue

elseif j <= i-2 % apenas uma transmissão bem-sucedida é possível num intervalo de tempo

matrizP(i,j)=0;

elseif j==i-1 % exatamente uma transmissão de utilizadores Ativos e não há chegadas de %mensagens de utilizadores Inativos

matrizP(i,j)=(i-1)\*(prob)\*((1-prob)^(i-2))\*((1-sigma)^(Nusers-(i-1)));

elseif j==i % nenhuma mensagem de chegada de utilizadores Inativos e zero ou mais %transmissões de utilizadores Ativos ou exatamente uma transmissão de utilizadores Ativos %e exatamente uma mensagem de chegada de utilizadores Inativos

matrizP(i,j)=((1-sigma)^(Nusers-(i-1)))\*(1-(i-1)\*(prob)\*((1-prob)^(i-2))) + (Nusers- 1)\*sigma\*((1-sigma)^(Nusers-(i-1)-1))\*(i-1)\*(prob)\*((1-prob)^(i-2));

elseif j>i % j-i mensagens de chegada de utilizadores Inativos e zero ou mais %transmissões de utilizadores Ativos ou j-i+1 mensagens de chegada de utilizadores %Inativos e %exatamente uma transimissão de utilizadores Ativos

matrizP(i,j)= nchoosek(Nusers-(i-1),(j-1)-(i-1))\* (sigma^((j-1)-(i-1)))\*((1- sigma)^(Nusers-(j-1)))\*(1-(i-1)\*(prob)\*((1-prob)^(i-2))) + nchoosek(Nusers-(i-1),(j- 1)-(i-1)+1)\* (sigma^((j-1)-(i-1)+1))\*((1-sigma)^(Nusers-(j-1)-1))\*(i-1)\*(prob)\*((1- prob)^(i-2));

end

j=j+1;

end

f=matrizP;

end

function [ f ] = matrizPI(prob, sigma, Nusers )

% Matriz com a Probabilidade PI para cada estado i (N) referente à matriz de

%transição de estados

% Matriz de Transicao dos Estados

mP = matrizTransicaoEstados(prob,sigma,Nusers);

% N\*N matriz com todos elementos iguais a 1

E=ones(Nusers);

% Matriz de identidade N por N com 1's na diagonal principal e zeros no resto da matriz

I=eye(Nusers);

% Matriz probabilidade PI, Limited State Probability

mPI = (ones(1,Nusers))\*((mP+E+I)^(-1));

f = mPI;

end

function [ f ] = theoreticalThroughput( prob, sigma, Nusers )

% Performance Slotted ALOHA - Theoretical Throughput value

% Probabilidade estacionária para cada estado i (N) referente à matriz de

%transição de estados

PI= matrizPI(prob, sigma, Nusers);

S=0; % Mean Throughput

% Contador

i=1;

while i<=Nusers

S = S + PI(1,i)\*conditionalThroughput(prob,i);

i=i+1;

end

f=S;

end

*(Para o exercício 1.b)*

function f = actualizacaoEstados(prob, sigma, users)

vector\_Users = users;

tamanho= size(users);

for i=1:tamanho(2)

if users(i)==1 % utilizador está no estado ativo

if Bernoulli(prob)==1 % utlizador tem pacote para enviar, espera para transmitir no %slot seguinte com probabilidade prob

vector\_Users(i)=2; % utlizador passa para estado de espera

end

elseif users(i)==0 % utilizador está no estado inativo

if Bernoulli(sigma)==1 % utilizador gerou um pacote para enviar com probabilidade %sigma

vector\_Users(i)=1; % utlizador passa para o estado ativo entao

end

end

end

f=vector\_Users; %vector dos N utilizadores com a atualização dos estados

end

function f = mudarEstado(users, estado)

vector\_Users=users;

tamanho=size(users);

for i=1:tamanho(2)

if estado==1 % estado ativo

if vector\_Users(i)==2 % Estava a espera de enviar pacote, houve colisão neste caso %porque esta no estado ativo

vector\_Users(i)=1; % Continua a espera de puder enviar pacote quando for possível, %fica no estado ativo

end

else % estado inativo

if vector\_Users(i)==2 % Estava a espera de enviar pacote

vector\_Users(i)=0; % Pacote enviado com sucesso

end

end

end

f=vector\_Users; %vector dos N utilizadores com a mudança dos estados

end

function f = numeroColisoes(users)

vector\_Users=users; % Vector de utilizadores

contador\_esperaUsers = 0;

tamanho=size(users);

i=1; % Variavel auxiliar

while i<tamanho(2) && contador\_esperaUsers<2

if vector\_Users(i)==2 % utilizador está no estado de espera para enviar pacote, aumenta %o número de utilizadores à espera

contador\_esperaUsers = contador\_esperaUsers + 1;

end

i=i+1;

end

f = contador\_esperaUsers; %número de utilizadores à espera para enviarem respetivos pacotes

end

function ThroughPut = slottedAlohaSimulation(prob, sigma, Nusers, Nslots)

%Estados existentes: 0 - inativo ; 1 - ativo ; 2 - espera

vector\_Users = zeros(1,Nusers);

numero\_sucessos = 0;

for i=1:Nslots

vector\_Users=actualizacaoEstados(prob, sigma, vector\_Users); % atualizacao de estados, %gera-se os pacotes a enviar para as probabilidades dadas para cada utilizador

if numeroColisoes(vector\_Users)>1 % Existiram colisoes no envio de pacotes

vector\_Users=mudarEstado(vector\_Users,1); % Utilizadores que sofreram de colisoes %ficam no estado ativo para puderem enviar depois

elseif numeroColisoes(vector\_Users)==1

vector\_Users=mudarEstado(vector\_Users,0); % Utiliadores que conseguiram depois enviar %pacote passam para estado inativo

numero\_sucessos=numero\_sucessos+1;

end

end

ThroughPut = numero\_sucessos/Nslots;

end

function [ f] = ex1( prob, sigma, Nusers, Nslots )

resultado\_pratico=slottedAlohaSimulation(prob,sigma,Nusers,Nslots);

resultado\_teorico=theoreticalThroughput(prob,sigma,Nusers);

disp('Theoretical throughput value');

disp(resultado\_teorico);

disp('Resultado obtido através da simulação');

disp(resultado\_pratico);

end

**Anexo exercício 2:**

function parameters

global LinkCapacity;

global Flows;

global endTime;

%USER DEFINED PARAMETERS%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Capacity of the link, in bits/sec

LinkCapacity=64000;

%Flows is a cell array where each cell corresponds to one flow, and each

%flow is a vector with 4 elements corresponding to (1) source type, (2)

%mean interarrival time (in seconds), (3) the mean packet length (in bits),

%and (4) priority level. There are two types of sources: 1 = Poisson

%arrivals and exponentially distributed sizes; 2 = Poisson arrivals and

%fixed sizes. The levels of priority must be consecutive integers starting

%at 1, where a lower number corresponds to a higher priority.

Flows={[1,1/16,1000,1];

[1,1/16,1000,1]};

%Definition of the simulation end time, function of the maximum mean

%interarrival time

endTime=1000\*(1/16);

**Anexo exercício 3:**

function parameters

global LinkCapacity;

global Flows;

global endTime;

%USER DEFINED PARAMETERS%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Capacity of the link, in bits/sec

LinkCapacity=64000;

%Flows is a cell array where each cell corresponds to one flow, and each

%flow is a vector with 4 elements corresponding to (1) source type, (2)

%mean interarrival time (in seconds), (3) the mean packet length (in bits),

%and (4) priority level. There are two types of sources: 1 = Poisson

%arrivals and exponentially distributed sizes; 2 = Poisson arrivals and

%fixed sizes. The levels of priority must be consecutive integers starting

%at 1, where a lower number corresponds to a higher priority.

Flows={[2,1/16,1000,1];

[2,1/16,1000,1]};

%Definition of the simulation end time, function of the maximum mean

%interarrival time

endTime=1000\*(1/16);

**Anexo exercício 4:**

function parameters

global LinkCapacity;

global Flows;

global endTime;

%USER DEFINED PARAMETERS%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Capacity of the link, in bits/sec

LinkCapacity=64000;

%Flows is a cell array where each cell corresponds to one flow, and each

%flow is a vector with 4 elements corresponding to (1) source type, (2)

%mean interarrival time (in seconds), (3) the mean packet length (in bits),

%and (4) priority level. There are two types of sources: 1 = Poisson

%arrivals and exponentially distributed sizes; 2 = Poisson arrivals and

%fixed sizes. The levels of priority must be consecutive integers starting

%at 1, where a lower number corresponds to a higher priority.

Flows={[2,1/16,1000,1];

[2,1/16,1000,2]};

%Definition of the simulation end time, function of the maximum mean

%interarrival time

endTime=1000\*(1/16);

**Anexo exercício 5:**

function parameters

global LinkCapacity;

global Flows;

global endTime;

%USER DEFINED PARAMETERS%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Capacity of the link, in bits/sec

LinkCapacity=64000;

%Flows is a cell array where each cell corresponds to one flow, and each

%flow is a vector with 4 elements corresponding to (1) source type, (2)

%mean interarrival time (in seconds), (3) the mean packet length (in bits),

%and (4) priority level. There are two types of sources: 1 = Poisson

%arrivals and exponentially distributed sizes; 2 = Poisson arrivals and

%fixed sizes. The levels of priority must be consecutive integers starting

%at 1, where a lower number corresponds to a higher priority.

Flows={[2,1/16,150,1,100];

[2,1/16,300,2,200]};

%Definition of the simulation end time, function of the maximum mean

%interarrival time

endTime=1000\*(1/16);

function init

global Time;

global EventList;

global Flows;

global numFlows;

global FlowStats;

global Queues;

global numQueues;

global numPacketsInQueues;

global TxLink;

global LinkState;

global Credits;

global currentQueue;

%Define que a próxima fila a ser servida. A 1ª a ser servida é a 1.

currentQueue = 1;

%Initialization of simulation clock

Time=0;

%Number of priority levels at the link

numFlows=size(Flows,1);

numPriorities=1;

for i=1:numFlows

Credits(i)= Flows{i}(5); %inicialização dos créditos disponíveis por queue.

if Flows{i}(4)>numPriorities

numPriorities=Flows{i}(4);

end

end

%Initialization of link data structures

numQueues=numPriorities; %One queue for each priority level

TxLink=[]; %Transmission link is empty

LinkState=0; %State of transmission link is idle

for i=1:numQueues

numPacketsInQueues(i)=0; %This queue is empty

Queues{i}=zeros(0,3); %Initialization of Queues

end

%Initialization of flow data structures

for i=1:numFlows

FlowStats{i}=[0,0,0]; %Statistics of this flow

nextEventType=1; %Next event type is arrival

MeanInterarrival=Flows{i}(2); %Mean interarrival time of this flow

nextArrivalTime=-MeanInterarrival\*log(rand()); %Arrival time of next packet of this %flow

EventList(i,:)=[nextArrivalTime i nextEventType]; %Schedules next packet arrival for &this flow

end

function pq

global Queues;

global numPacketsInQueues;

global TxLink;

global LinkState;

global LinkCapacity;

global EventList;

global Time;

global numQueues;

global currentQueue;

global Flows;

global Credits;

thisQueue=0;

if(currentQueue > numQueues)

currentQueue=1; %verfica se já foi dada uma volta completa

end

firstQueue=currentQueue; %garante que o ciclo não fica num loop infinito

noPackets=0;

while(1)

if isempty(Queues{currentQueue})

%nao faz nada pq o ponteiro e actualizado no fim

else

pacote = Queues{currentQueue}(1,:);

if (Credits(currentQueue)>= pacote(3))

Credits(currentQueue)= Credits(currentQueue) - pacote(3);

break;

else

Credits(currentQueue)= Credits(currentQueue)+ Flows{currentQueue}(5); %actualizar %o conter de creditos

end

end

currentQueue = currentQueue + 1; %actualizar para proxima fila

if(currentQueue > numQueues) %averigua se é preciso voltar ao início - Round Robin

currentQueue = 1;

end

if(firstQueue == currentQueue) %averigua se já foi dada uma volta completa

noPackets = 1; %nao há pacotes a enviar, o scheduler não envia nada.

break;

end

end

thisQueue=currentQueue; %queue que a ser enviada

currentQueue=currentQueue+1; %actualiza a currentQueue;

if(noPackets==0) %apenas envia se houver um pacote por enviar

%Transfers selected packet to transmission link

thisPacket=Queues{thisQueue}(1,:);%Reads packet to be transmitted

Queues{thisQueue}(1,:)=[]; %Removes selected packet from this queue

numPacketsInQueues(thisQueue)=numPacketsInQueues(thisQueue)-1; %Decrements number of %packets in this queue

TxLink=thisPacket; %Stores this packet at the transmission link

LinkState=1; %Set state of link to busy

%Schedules departure of this packet

thisLength=thisPacket(3); %Length of this packet

nextDepartureTime=Time+thisLength/LinkCapacity; %Time of next departure

nextEventType=2; %Next event type is departure

EventList(end+1,:)=[nextDepartureTime 0 nextEventType]; %Places departure event in %event list

end

end

***Nota:*** Relativamente aos exercícios 2-5, as funções que não foram aqui apresentado em anexo mas que são utilizadas para a concretização das simulações apresentadas, encontram-se inalterado comparativamente às funções fornecidas no *fenix*, logo não são apresentadas em anexo.